# Challenges in Validating FLOSS Conguration

Markus Raab, Gergö Barany

# Challenges in Validating FLOSS Configuration

Markus Raab[1] and Gergö Barany[2⋆]

[1] Institute of Computer Languages, Vienna University of Technology, Austria
markus.raab@complang.tuwien.ac.at
[2] Inria Paris, France
gergo.barany@inria.fr

**Abstract.** Developers invest much effort into validating configuration during startup of free/libre and open source software (FLOSS) applications. Nevertheless, hardly any tools exist to validate configuration files to detect misconfigurations earlier. This paper aims at understanding the challenges to provide better tools for configuration validation. We use mixed methodology: (1) We analyzed 2,683 run-time configuration accesses in the source-code of 16 applications comprising 50 million lines of code. (2) We conducted a questionnaire survey with 162 FLOSS contributors completing the survey. We report our experiences about building up a FLOSS community that tackles the issues by unifying configuration validation with an external configuration access specification.
We discovered that information necessary for validation is often missing in the applications and FLOSS developers dislike dependencies on external packages for such validations.

## 1 Introduction

*Configuration settings* influence the behavior of software and are used ubiquitously today. *Configuration access* is done by the part of applications concerned with fetching configuration settings from configuration files, environment variables, etc. at run-time. *Configuration validation* detects configuration settings which do not fulfill the user's expectations, for example, setting a web browser's proxy to a server that is not reachable in the currently connected network.

While configuration access seems to be straightforward, system administrators experience many surprises on a daily basis. In the systems community the issue is well-known as *misconfiguration* [37,30,1,36]. Misconfigurations cause large-scale outages of Internet services [19]. Yin et al. [37] claim that *"a majority of misconfigurations (*70.0 %∼85.5 %*) are due to mistakes in setting configuration"*.

Xu et al. argue that often configuration access code and not system administrators are to blame [35]. Often (38.1 %∼53.7 %) misconfiguration is caused by illegal settings which clearly violate syntactic or semantic rules [37]. Thus most errors could be caught with a consistency checker executed before configuration changes. Nevertheless, only in 7.2 % to 15.5 % cases do error messages pinpoint the error [37]. Free/libre and open source software (FLOSS) applications often

⋆ This work was performed while the author was at CEA LIST Software Reliability Laboratory, France, and supported by the French National Research Agency (ANR), project AnaStaSec, ANR-14-CE28-0014.

do not validate their settings before startup or even later [34]. System administrators have to find their own ad-hoc ways [3,4,13,31,39].

Other factors also influence configuration settings. We will call validation that considers more than the settings of a single application *global validation*. Faulty global validation causes issues in 46.3 %∼61.9 % of cases [37]. For example, when a web browser is started in a different network, previously working proxy settings will fail to work. Our holistic approach rejects misconfigurations early on.

These issues lead to our **research question**: Why do we lack tools for global validation, and how can we help developers provide them?

Our contributions are as follows:

- We showed that `getenv` is omnipresent and popular (Section 3).
- We unveiled challenges related to current configuration systems (Section 4).
- We implemented a tool implementing the unearthed requirements (Section 5).
- The tool is available as free software at `https://www.libelektra.org`.

## 2 Methodology

Our methodological foundation builds on "theory building from cases" [10,11]. In the present paper we will use two different methodologies embedded in a framework: source-code analysis and a questionnaire.

### 2.1 Source-code Analysis

We study `getenv`, which is an application programming interface (API) to access environment variables. We chose it because it is the only widely standardized configuration access API (including C, C++, and POSIX) and available in many programming languages. In earlier work [26], we showed that `getenv` is used at run-time ubiquitously. `getenv` is often combined with other techniques (e. g., config files), sometimes overriding configuration file settings. Furthermore, environment variables are not part of configuration settings dialogues, i. e., they are certainly not validated.

We carefully selected 16 applications across different domains. We included large applications with a thriving community but also others for diversity. We used the versions of the applications as included in Debian 8 (Jessie) as shown later in Table 1. We downloaded package sources from `http://snapshot.debian.org`. To determine the code size we used Cloc 1.60 [7].

We manually counted all `getenv` occurrences for the version specified in Table 1. Then we categorized the resulting 2,683 code snippets around `getenv`. We looked if `getenv` occurrences *depend* on some other configuration. Such situations occur when configuration settings interact; for example, fallback chains of configuration access points depend on each other. Such fallback chains are hints to global configuration access, which we wanted to find. As our last experiment, we searched for places where global validation would be useful, and investigated how helpful the documentation of the `getenv` parameters is.

**Threats to Validity:** For evaluating usefulness (as only done in the last experiment), by nature, subjectivity is involved. In particular, it is possible that we overlooked dependences. We will report the numbers we found but we consider the experiment as exploratory and not as something that could be repeated with the same numbers as outcome. The individual examples, however, are insightful.

### 2.2 Questionnaire

We carefully prepared a questionnaire with FLOSS developers in mind. Then we conducted pilot surveys with developers, colleagues and experts for surveys. In the iterations we improved the questions and made the layout more appealing.

In order to reach the target group, we posted requests to fill out the survey in the respective FLOSS communication channels. To obtain a higher quality, we awarded non-anonymous answers with small donations to FLOSS-related projects. We used the non-anonymous answers to cross-check statistics.

We asked some personal questions about age, education, occupation, and FLOSS participation to have the important characteristics of our participants.

We used Limesurvey version 2.50+ for conducting the survey. We will report the percentages relative to the number of persons ($n$) who answered a particular question. We report means and standard deviations ($s$) of samples for $n \geq 95$. We used the Kolmogorov-Smirnov test [15] for smaller samples.

**Threats to Validity:** For the validity of our survey it is important that only FLOSS contributors participate. The donation might have persuaded some participants to fill out parts of the survey even though they had no particular experience. Thus we explicitly asked about contributions to specific projects.

The survey reflects the beliefs of participants. Thus we used other methods to distill facts about the applications. Because opinions help to understand goals and reasons, the survey is an important part of the overall study. It should be considered as supplement to the source-code analysis.

**Demographics:** The front page of the survey was shown to 672 persons, 286 gave at least one answer, 162 completed the questionnaire, and 116 persons entered their email addresses. The age of the population ($n = 220$) has a mean of 32 years ($s = 9$). The degrees in the population ($n = 244$) are: master (38 %), bachelor (25 %), student (18 %), no degree (13 %), or PhD (6 %). As their occupation, 56 % of the persons selected software developer, 21 % system administrator, and 16 % researcher (multiple choice question, $n = 287$). Participants reported work on up to five different FLOSS projects. For the first project, they estimated their participation with a mean of 5.3 years ($s = 5$, $n = 180$). 60 % of them reported a second FLOSS project, 36 % a third, 17 % a fourth, and 9 % a fifth.

Raw data and questions are available at `https://rawdata.libelektra.org`.

## 3 Configuration Access

Before we start exploring our research question, we need to validate that our evaluated configuration accesses are indeed relevant and popular. In this section we investigate which configuration access methods FLOSS developers use.

### 3.1 Which Methods for Configuration Access are Popular?

**Finding** $1_a$**:** *We observed that* `getenv` *is omnipresent with 2,683 occurrences.*

The source code of the applications we analyzed has 4,650 textual `getenv` occurrences. 2,683 of them were actual `getenv` invocations, 1,967 were occurrences in comments, ChangeLog, build system, or similar. (See Table 1 for details.)

**Finding** $1_b$**:** *Three kinds of configuration access are equally popular: Command-line arguments, environment variables, and configuration files. Developers are highly satisfied with them. Others are used less and less liked.*

Command-line arguments (92 %, $n = 222$), environment variables (e.g., via `getenv`) (79 %, $n = 218$), and configuration files (74 %, $n = 218$) are the most popular ways to access configuration. Other systems, such as X/Q/GSettings, KConfig, dconf, plist, or Windows Registry, were used less ($\leq 13$ %, $n \geq 185$).

Participants rarely found it (very) frustrating to work with the popular systems: `getenv` (10 %, $n = 198$), configuration files (6 %, $n = 190$), and command-line options (4 %, $n = 210$). Less-used systems frustrated more ($\geq 14$ %, $n \geq 27$).

### 3.2 What is the Purpose of `getenv`?

**Finding** $1_c$**:** *Like other configuration accesses,* `getenv` *is used to access configuration settings (57 %). Sometimes it bypasses main configuration access.*

Of the 2,683 `getenv` invocations, 1,531, i. e., 57 %, relate to run-time configuration settings and not debugging, build-system, or similar. Further investigations in this paper elaborate on these 1,531 `getenv` occurrences.

We found occurrences where `getenv` obviously bypasses main configuration access, for example, to configure the location of configuration files.

Also in the survey we asked about the purpose of `getenv` ($n = 177$). The reasons to use it vary: in a multiple choice question 55 % say they would use it for debugging/testing, 45 % would use `getenv` to bypass the main configuration access, and 20 % would use `getenv` if configuration were unlikely to be changed.

**Finding** $1_d$**:** *In many cases* `getenv` *parameters are shared between applications.*

In the source code we investigated which parameters were passed to `getenv`. We found that 716 parameters were shareable parameters such as `PATH`. In the survey 53 % say they use `getenv` for configuration integration ($n = 177$).

**Finding** $1_e$**:** *Parameters of* `getenv` *are often undocumented.*

The function parameter passed to `getenv` invocations tells us which configuration setting is accessed. In an Internet search using the application's and `getenv` parameter's name with `startpage.com`, we found documentation for only 283 of the non-shared `getenv` parameters but not for the 387 others.

The FLOSS projects deal with the missing documentation of `getenv` parameters in different ways. Most projects simply claim their `getenv` usage as internal, saying the environment should not be used for configuration by end users, even if there is no other way of achieving some goal. Often we miss a specification describing which parameters are available.

In some other projects, the developers invest effort to create lists of available parameters. For example, in Libreoffice developers try to find `getenv` occurrences automatically with `grep`, which fails with `getenv` aliases[3].
**Discussion:** The `getenv` API has some severe limitations and is sometimes a second-class citizen. One limitation is that return values of `getenv` invocations cannot be updated by other processes. For example, `getenv("http_proxy")` within a running process will still return the old proxy, even if the user changed it elsewhere. Nevertheless, `getenv` supports all characteristics of configuration access and can be used to investigate challenges in configuration validation.
**Implication:** Configuration APIs should avoid returning outdated values. Environment variables, however, are no replacement for configuration files, thus they do not support persistent changes by applications. There is currently no satisfactory solution in FLOSS for global, shareable configuration settings.

## 4 Configuration Validation

Having established which configuration accesses are popular (including, but not limited to, `getenv`), we will investigate challenges of configuration validation.

### 4.1 Which are the Concerns Regarding Global Validation?

**Finding $2_a$:** *Developers have concerns about adding dependencies for global validation (84 %) and reducing configuration (30 %) but desire good defaults (80 %).*

Many persons (30 %, $n = 150$) think that the number of configuration settings should *not* be reduced. But 43 % said it should be reduced to prevent errors.

We got mixed answers ($n = 177$) to the question "Which effort do you think is worthwhile for providing better configuration experience?" Most persons (80 %) agree that proper defaults are important. Most methods exceed the effort considered tolerable by the majority of participants: Only `getenv` would be used by the majority (53 %). System APIs would be used by 44 %. Fewer (30 %) would use OS-specific data. Only 21 % of the participants would use dedicated libraries, 19 % parse other configuration files, and 16 % use external APIs that add new dependencies.
**Discussion:** To avoid dependencies, FLOSS developers currently expect users to configure their applications to be consistent with the global configuration.
**Implication:** The results indicate demand for dependency injection to have global validation without direct dependencies.

### 4.2 Which Challenges Prevent us from Supporting Validation?

**Finding $2_b$:** *Present configuration validation is encoded in a way unusable for external validation or introspection tools.*

In none of the 16 applications was the validation code kept separately, e.g., in a library. Instead it was scattered around like other cross-cutting concerns.

[3] `https://bugs.documentfoundation.org/show_bug.cgi?id=37338`

**Finding $2_c$:** *Developers are unable to support global validation, even if the problem is well-known and they put effort into it. We found out that information essential to check or fix constraints is not available within the applications.*

In Table 1 we present a list of applications and the `getenv` occurrences we analyzed. The column *textual getenv* contains the number of literal `getenv` occurrences, as we used it for the analysis of the histories. The column *lines per getenv* shows how often manually counted `getenv` occurs in code. The column *depend getenv* presents manually counted `getenv` occurrences that depend on, or are used by, other configuration code.

| application | version | 1k lines of code | textual getenv | counted getenv | lines per getenv | depend getenv |
|---|---|---|---|---|---|---|
| 0ad | 0.0.17 | 474 | 114 | 55 | 8,617 | 43 |
| Akonadi | 1.13.0 | 37 | 16 | 13 | 2,863 | 6 |
| Chromium | 45.0.2454 | 18,032 | 1,213 | 770 | 23,418 | 281 |
| Curl | 7.38.0 | 249 | 278 | 53 | 4,705 | 25 |
| Eclipse | 3.8.1 | 3,312 | 114 | 40 | 82,793 | 23 |
| Evolution | 3.12.9 | 673 | 33 | 23 | 29,252 | 5 |
| GCC | 4.9.2 | 6,851 | 727 | 377 | 18,172 | 143 |
| Firefox | 38.3.0esr | 12,395 | 1,052 | 788 | 15,730 | 271 |
| Gimp | 2.8.14 | 902 | 80 | 56 | 16,102 | 21 |
| Inkscape | 0.48.5 | 480 | 27 | 19 | 25,255 | 13 |
| Ipe | 7.1.4 | 116 | 29 | 21 | 5,529 | 14 |
| Libreoffice | 4.3.3 | 5,482 | 427 | 284 | 19,304 | 143 |
| Lynx | 2.8.9dev1 | 192 | 112 | 89 | 2,157 | 66 |
| Man | 2.7.0.2 | 142 | 248 | 62 | 2,293 | 42 |
| Smplayer | 14.9.0 ds0 | 76 | 1 | 1 | 76,170 | 1 |
| Wget | 1.16 | 143 | 179 | 32 | 4,456 | 18 |
| Total | | 49,556 | 4,650 | 2,683 | 18,470 | 1115 |
| Median | | 477 | 114 | 54 | | 24 |

**Table 1.** Automatic and manual count of getenv occurrences.

Most of these places (1115, i. e., 73 %) were dependent on some other configuration. We found 204 places where some kind of configuration dependencies were forgotten. In 58 cases we found several hints, e.g., fallback chains with missing cases or complaints on the Internet about the not considered dependency.

We give a real-life example from the Eclipse source of how easily dependencies are forgotten. The missing dependencies lead to missing validation, which leads to frustrating user experience. If Eclipse wants to instantiate an internal web browser, some users get an error saying that `MOZILLA_FIVE_HOME` is not set. On GitHub alone, issues mentioning the variable were reported 71 times. The underlying issue usually is that the software package `webkitgtk` is missing[4]. The

[4] `https://groups.google.com/forum/#!topic/xmind/5SjPTy0MmEo`

developers even considered the dependency (installation paths) for RPM-based systems by parsing `gre.conf`. But for other users on non-RPM-based systems the fallback is to query `MOZILLA_FIVE_HOME` which leads to the misleading error. In Eclipse the workarounds (including parsing `gre.conf`) needed 213 lines of code. Furthermore, most of the 9006 code snippets we found on GitHub referring to the variable are small wrappers trying to set `MOZILLA_FIVE_HOME` correctly.
**Discussion:** While the package managers easily answer where the missing files are located, within the application there is no reasonable way to find out. We found similar issues concerning network, firewall, hardware settings, etc.
**Implication:** Applications have a need to access global configuration settings.

## 5 Experience Report on Supporting Global Validation

*Elektra* is a library that aims at providing unified access to configuration settings as key/value pairs. It integrates a specification language for global validation. Here we will discuss how Elektra fulfills the requirements unearthed by the study before describing the challenges to adoption that Elektra faced in the past.

### 5.1 Unify Configuration

We summarize requirements for configuration tools derived from the findings of Section 3.
**Requirement $1_{a-c}$:** *Developers use different mechanisms for configuration accesses interchangeably or to bypass limitations of others.* To avoid the need for bypasses, Elektra bootstraps itself at startup, making it possible for configuration settings to describe the configuration access, e. g., which configuration files should be used. To allow administrators to use all popular techniques, Elektra reads from different sources such as configuration files, system settings, and environment variables. Elektra integrates many different configuration file formats such as INI, XML, etc., and it supports notifications to always keep application's configuration settings in sync with the persistent configuration settings.
**Requirement $1_d$:** *FLOSS developers demand a way to share configuration settings.* We implemented a layer similar to a virtual file system, which enables applications and system administrators to mount configuration files [29]. This technique enables applications to access configuration settings of any other application. Using links and transformations [22] one can even configure applications to use other other settings without any support from the application itself.
**Requirement $1_e$:** *There should be a way to document configuration settings.* Elektra introduces specifications for configuration settings [24]. These specifications should also include documentation. But even if they do not, users at least know which configuration settings exist, and which values are valid for them.

### 5.2 Validate Configuration

**Requirement $2_a$:** *Dependencies exclusively needed for configuration settings should be avoided.* Elektra introduces plugins that enable a system-level depen-

dency injection. Developers specify validations in specifications, without the need for their application to depend on additional external libraries. In plugins executed on configuration access, the settings get validated or default settings get calculated. Elektra only uses the C standard library and no other external dependencies [20]. Nevertheless, even the dependency on Elektra itself can be avoided. Elektra supports intercepting of library calls such as `getenv` [26,27]. Using this technique, applications think they use environment variables, while in reality they query Elektra.

**Requirement $2_b$:** *Configuration settings and validations should be open to introspection.* Similarly to `getenv`, Elektra provides an API, but it aims to overcome the limitations of previous abstractions and interfaces. Elektra allows many configuration files to be integrated with a uniform key/value API. Even the specifications of accesses, dependencies, and validations are accessible via the same API. Thus system administrators and applications can use the API to introspect configuration settings. Applications need the permission to open the configuration files.

**Requirement $2_c$:** *Global validation should be supported.* Elektra supports global validation through a range of different checker plugins. These plugins do not only check data for consistency but also check if configuration settings conflict with reality. For example, one plugin checks for presence of files or directories, while another plugin checks if a host name can be resolved. Checks are executed whenever Elektra's API is used for writing. This way also all administrator tools sitting on top of Elektra reject invalid configuration settings. Elektra also allows to integrate system information such as hardware settings via plugins.

### 5.3 Community Building

The *Elektra Initiative* is a community that started with the straightforward idea to have a single API for configuration access. Other projects watched how it progressed, but adoptions occurred rarely. Due to various grave issues in the first versions, the API needed several redesigns. Obviously, API changes are not very popular and Elektra lost most of its users at this time. Despite many marketing efforts to change the situation, it was predominantly companies and not FLOSS software that used Elektra. This slow adoption was unfortunate but an opportunity to continue making changes. Unfortunately, the changes were not done wisely, instead we introduced mostly unnecessary features. Here the Elektra Initiative had its low and turning-point.

Then the goals shifted towards a more technical solution: We avoid marketing campaigns to persuade everyone to use the API with arguments like "it will give benefits once everyone migrates" but instead it should offer immediate advantages previous APIs did not have. This meant Elektra went into a highly competitive market facing the difficulty of being better than any other configuration library. As a distinctive feature, we started to aim for global validation but without giving up compatibility to current configuration files.

These changes made the core more complicated, which led to a recruiting problem. The documentation was for a long time only a master thesis [20], which

was a very unsatisfactory situation. The next efforts were to make the project community-friendly again. We started to improve quality by regression tests, fixing reports of code analysis tools, and adding code comments and assertions. Then we started overhauling documentation, writing tutorials, and created a website. Last but not least, we started releasing every two months with fixes and new features in every release. These changes led to more than a dozen contributors, Elektra being packaged for many distributions, and acceptance of a paper on Elektra in "The Journal of Open Source Software" [23].

## 6 Community Feedback and Future Work

The survey validated Elektra's goals: Many agreed (80 %, $n = 153$) that a solution must be lightweight and efficient; and that a configuration library must be available anywhere and anytime (84 %, $n = 153$). Many persons (70 %, $n = 150$) consider it important that the community is supportive. Even more persons want bugs to be fixed promptly (88 %, $n = 150$). Because 76 % persons find it important that applications directly ship documentation ($n = 157$), external specifications should have documentation. Nearly everyone (96 %, $n = 173$) agrees that configuration integration, such as global validation, would at least moderately improve user experience. Thus we will continue research in this area.

A participant said: *"Must be extensible/adaptable. If it is, users can take care of many of the above aspects themselves"*. We agree and continue to pioneer modularity. For example, many persons found readability of configurations important (65 %, $n = 152$) but could not agree which formats are readable.

Another person wrote: *"It must offer a compelling reason to switch from e.g gsettings. For example a killer feature that others don't have, etc. Otherwise, the status quo wins."* Elektra's "killer feature" can be global validation.

From our experience with Elektra, it was also clear that we need to put much more effort into API stability. Thus we avoid breaking changes to the API. We are about to provide easy-to-use high-level APIs for different use cases.

The 1.0 release of Elektra is still pending: (1) The specification language for validation/transformation/dependency injection is not completely defined. (2) The configuration parsers have limitations, e. g., they do not always preserve comments or order. (3) Elektra puts some unnecessary limitations on the plugins.

## 7 Related Work

Many other configuration libraries have validation capabilities, for example, Apache Commons Configuration. Unlike Elektra they do not have external specifications. Instead they require developers to hardcode them into the applications.

Other papers describe the technical details of Elektra [29,20,23]. In particular frontend code generation avoids errors in configuration access [28,21]. Other work describes Elektra's specification language [22,24] and how applications participate without code modifications [25,26].

Crowston et al. [6] created a survey of empirical research for FLOSS. Michlmayr et al. [16] investigated quality issues of FLOSS using interviews. We were able to confirm that documentation often is lacking. Barcomb et al. [2] used a questionnaire to investigate how developers acquire FLOSS skills.

PCheck [34] validates configuration files early. Unlike Elektra, it is not free software and does not support application-specific checks or plugins. Some work was done to automatically resolve misconfiguration [33,30,1,38]. These approaches aim at solving already manifested issues, Elektra aims at resolving them earlier. Xu et al. [36] surveyed further possibilities.

Nosál et al. [17,18] investigated similar abstractions but with a focus on language mapping. Denisov [8] collected requirements for configuration libraries.

Berger et al. [5] and Villela et al. [32] created a questionnaire that asks about variability modeling. Our survey focused on a different target group.

## 8 Conclusions

In this paper we examined challenges in configuration access and presented a solution. We addressed the research question: Why do we lack tools for global validation and how can we help developers provide them? The answer is that validations are encoded in the software in a way (1) unusable by external tools, and (2) incapable of using global knowledge of the system. The answer is backed up by both a questionnaire and a source analysis.

To overcome developers' configuration issues, we need to externalize configuration access specifications and use a unified configuration library. The empirical data backs up that this is possible and wanted. It is *possible*, because currently different configuration accesses are used interchangeably. It is *wanted*, because users stated that different forms of configuration access sources should be able to override each other.

Based on our survey we might have to rethink how to reduce the number of configuration settings because many developers do not agree with complete removal of less-used settings. The survey also showed that external dependencies in configuration access code are a contradictory topic: Developers want good defaults, but do not want to pay for them with dependencies. Elektra's way of implementing dependency injection and globally calculating default settings fulfills both goals. Because of the externalization of configuration access specifications, users can even introspect the (default) settings that applications receive.

Finally, we described FLOSS community efforts to improve on the issues. The results show that a dependency injection at the system level is feasible and practical. It has the potential to be accepted by developers if they perceive global integration and validation as "killer feature". The current status of the FLOSS project can be tracked at `https://www.libelektra.org`.

**Acknowledgements:** We thank the anonymous reviewers, Tianyin Xu, Franz Puntigam, Stefan Winter, Milan Nosál, and Harald Geyer for detailed reviews of this paper. Additionally, many thanks to all the people contributing to Elektra.

## References


1. M. Attariyan and J. Flinn. Automating configuration troubleshooting with dynamic information flow analysis. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI'10, pages 1–11, Berkeley, CA, USA, 2010. USENIX Association.
2. A. Barcomb, M. Grottke, J.-P. Stauffert, D. Riehle, and S. Jahn. How developers acquire FLOSS skills. In *IFIP International Conference on Open Source Systems*, pages 23–32. Springer, 2015.
3. R. Barrett, Y.-Y. M. Chen, and P. P. Maglio. System administrators are users, too: designing workspaces for managing Internet-scale systems. In *CHI'03 Extended Abstracts on Human Factors in Computing Systems*, pages 1068–1069. ACM, 2003.
4. R. Barrett, E. Kandogan, P. P. Maglio, E. M. Haber, L. A. Takayama, and M. Prabaker. Field studies of computer system administrators: analysis of system management tools and practices. In *Proceedings of the 2004 ACM conference on Computer supported cooperative work*, pages 388–395. ACM, 2004.
5. T. Berger, R. Rublack, D. Nair, J. M. Atlee, M. Becker, K. Czarnecki, and A. Wąsowski. A survey of variability modeling in industrial practice. In *Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems*, VaMoS '13, pages 7:1–7:8, New York, NY, USA, 2013. ACM.
6. K. Crowston, K. Wei, J. Howison, and A. Wiggins. Free/libre open-source software development: What we know and what we do not know. *ACM Comput. Surv.*, 44(2):7:1–7:35, Mar. 2008.
7. A. Danial. Cloc–count lines of code. `https://github.com/AlDanial/cloc`, 2017. Accessed Februar 2017.
8. V. S. Denisov. Functional requirements for a modern application configuration framework. *International Journal of Open Information Technologies*, 10:6–10, 2015.
9. R. Di Cosmo, S. Zacchiroli, and P. Trezentos. Package upgrades in foss distributions: Details and challenges. In *Proceedings of the 1st International Workshop on Hot Topics in Software Upgrades*, HotSWUp '08, pages 7:1–7:5. ACM, 2008.
10. S. Easterbrook, J. Singer, M.-A. Storey, and D. Damian. Selecting empirical methods for software engineering research. In F. Shull, J. Singer, and D. Sjøberg, editors, *Guide to Advanced Empirical Software Engineering*, pages 285–311. Springer, 2008.
11. K. M. Eisenhardt and M. E. Graebner. Theory building from cases: Opportunities and challenges. *Academy of management journal*, 50(1):25–32, 2007.
12. R. A. Ghosh, R. Glott, B. Krieger, and G. Robles. Free/libre and open source software: Survey and study. International Institute of Infonomics, University of Maastricht, The Netherlands, 2002.
13. E. M. Haber and J. Bailey. Design guidelines for system administration tools developed through ethnographic field studies. In *Proceedings of the 2007 Symposium on Computer Human Interaction for the Management of Information Technology*, CHIMIT '07, New York, NY, USA, 2007. ACM.
14. I. Hammouda and M. Harsu. Documenting maintenance tasks using maintenance patterns. In *Eighth European Conference on Software Maintenance and Reengineering, 2004. CSMR 2004. Proceedings.*, pages 37–47, March 2004.
15. H. W. Lilliefors. On the Kolmogorov-Smirnov test for normality with mean and variance unknown. *Journal of the American Statistical Association*, 62(318):399–402, 1967.
16. M. Michlmayr, F. Hunt, and D. Probert. Quality practices and problems in free software projects. In *Proceedings of the First International Conference on Open Source Systems*, pages 24–28, 2005.

17. M. Nosál' and J. Porubän. Supporting multiple configuration sources using abstraction. *Open Computer Science*, 2(3):283–299, 2012.
18. M. Nosál and J. Porubän. XML to annotations mapping definition with patterns. *Computer Science and Information Systems*, 11(4):1455–1477, 2014.
19. D. Oppenheimer, A. Ganapathi, and D. A. Patterson. Why do Internet services fail, and what can be done about it? In *USENIX Symposium on Internet Technologies and Systems*, volume 67. Seattle, WA, 2003.
20. M. Raab. A modular approach to configuration storage. *Master's thesis, Vienna University of Technology*, 2010.
21. M. Raab. Global and thread-local activation of contextual program execution environments. In *Proceedings of the IEEE 18th International Symposium on Real-Time Distributed Computing Workshops (ISORCW/SEUS)*, pages 34–41, April 2015.
22. M. Raab. Sharing software configuration via specified links and transformation rules. In *Technical Report from KPS 2015*, volume 18. Vienna University of Technology, Complang Group, 2015.
23. M. Raab. Elektra: universal framework to access configuration parameters. *The Journal of Open Source Software*, 1(8), dec 2016.
24. M. Raab. Improving system integration using a modular configuration specification language. In *Companion Proceedings of the 15th International Conference on Modularity*, MODULARITY Companion 2016, pages 152–157. ACM, 2016.
25. M. Raab. Persistent contextual values as inter-process layers. In *Proceedings of the 1st International Workshop on Mobile Development*, Mobile! 2016, pages 9–16. ACM, 2016.
26. M. Raab. Unanticipated context awareness for software configuration access using the getenv API. In *Computer and Information Science*, pages 41–57. Springer International Publishing, Cham, 2016.
27. M. Raab and G. Barany. Introducing context awareness in unmodified, context-unaware software. In *12th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE)*, 2017. To appear.
28. M. Raab and F. Puntigam. Program execution environments as contextual values. In *Proceedings of 6th International Workshop on Context-Oriented Programming*, pages 8:1–8:6, New York, NY, USA, 2014. ACM.
29. M. Raab and P. Sabin. Implementation of Multiple Key Databases for Shared Configuration. `ftp://www.markus-raab.org/elektra.pdf`, Mar. 2008. Accessed February 2014.
30. Y.-Y. Su, M. Attariyan, and J. Flinn. Autobash: improving configuration management with operating system causality analysis. *ACM SIGOPS Operating Systems Review*, 41(6):237–250, 2007.
31. N. F. Velasquez, S. Weisband, and A. Durcikova. Designing tools for system administrators: An empirical test of the integrated user satisfaction model. In *Proceedings of the 22nd Conference on Large Installation System Administration Conference*, LISA'08, pages 1–8, Berkeley, CA, USA, 2008. USENIX Association.
32. K. Villela, A. Silva, T. Vale, and E. S. de Almeida. A survey on software variability management approaches. In *Proceedings of the 18th International Software Product Line Conference - Volume 1*, SPLC '14, pages 147–156. ACM, 2014.
33. H. J. Wang, J. C. Platt, Y. Chen, R. Zhang, and Y.-M. Wang. Automatic misconfiguration troubleshooting with peerpressure. In *OSDI*, volume 4, pages 245–257, 2004.

34. T. Xu, X. Jin, P. Huang, Y. Zhou, S. Lu, L. Jin, and S. Pasupathy. Early Detection of Configuration Errors to Reduce Failure Damage. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI'16)*, Nov. 2016.
35. T. Xu, J. Zhang, P. Huang, J. Zheng, T. Sheng, D. Yuan, Y. Zhou, and S. Pasupathy. Do not blame users for misconfigurations. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 244–259. ACM, 2013.
36. T. Xu and Y. Zhou. Systems approaches to tackling configuration errors: A survey. *ACM Comput. Surv.*, 47(4):70:1–70:41, July 2015.
37. Z. Yin, X. Ma, J. Zheng, Y. Zhou, L. N. Bairavasundaram, and S. Pasupathy. An empirical study on configuration errors in commercial and open source systems. SOSP '11, pages 159–172, 2011.
38. S. Zhang and M. D. Ernst. Automated diagnosis of software configuration errors. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 312–321, Piscataway, NJ, USA, 2013. IEEE Press.
39. S. Zhang and M. D. Ernst. Which configuration option should I change? In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 152–163, New York, NY, USA, 2014. ACM.